Mining Logic Models in the Presence of Noisy Data

Giovanni Felici and Emanuel Weitschek
Istituto di Analisi dei Sistemi ed Informatica
Consiglio Nazionale delle Ricerche
Viale Manzoni 30
Roma, Italy 00185
{giovanni.felici, emanuel.weitschek}@iasi.cnr.it

Abstract

In this work we consider a method for the extraction of knowledge expressed in Disjunctive Normal Form (DNF) from data. The method is mainly designed for classification purposes, and is based on three main steps: Discretization, Feature Selection, and Formula Extraction. The three steps are formulated as optimization problems and solved with ad hoc algorithmic strategies. When used for classification purposes, the proposed approach is designed to perform exact separation of training data and can thus be exposed to overfitting when a significant amount of noise is present. We analyze the main problems that may arise when this method deals with noisy data and propose extensions for the three steps of the method.

Introduction

Logic Data Mining is characterized by the use of a logic description of the data and of a logic description of the classification and explanatory model that is extracted from data.

Each observed object of the training set is described by a set of logic variables - e.g., variables that may assume the value *True* or *False*. The extraction of knowledge from data may often be affected by the presence of *noise* in the observed data. Noise can appear in different forms, and often its definition depends on the type of model and analysis adopted; here we assume a very general definition, stating that noise in data can appear in three different ways: 1) elements of the training data that do not effectively belong to the universe that we are studying; 2) a feature that is not related with the class that we are willing to predict; 3) some values of the features that have been wrongly observed or measured.

One of the main assumption in data mining and data analysis is that a certain model is present in the observed data, and the aim of these techniques is to uncover such model. The three types of noise mentioned above can make this task much harder. One typical sign of the presence of noise is *overfitting*, that may take place when the model extracted from training data is biased by noise and thus it is not able to reproduce a correct behavior on test data.

Dealing with noise is a necessity for all methods, but it acquires a particular relevance in the case of Logic Data Mining methods, where even small errors may cause the flipping of a variable from *True* to *False*; and it is even more important for those methods that do not pose restrictions on the complexity of the explanatory model sought for in the data, as it is the case for the methodology described in this paper.

In the following sections we provide some introductory material, a brief description of the Logic Data Mining framework that is proposed, and a description of the new extensions designed to deal with the problem of noisy data.

Classification and Data Mining

Classification is the assignment of a new object into a predefined class after examining its characteristics. It is an important Data Mining activity, where the value of a variable, usually named *class variable*, is predicted on the basis of some independent variables (or features). The task of a classification method is to identify and extract the model that relates the class variable to the features of the elements of a training set. Examples include separating healthy patients from diseased ones on the basis of their clinical records, or categorizing tumoral cells as benign or malignant based on their physical characteristics.

Besides the classification task just described, it is also of major relevance to provide good *explanations* of the classification model; for example to know what are the genes that are over-expressed in sick patients with respect to healthy ones would not just help in classifying new patients - probably a secondary task in medical diagnostics - but also in designing new experiments, therapies, and drugs to advance research and medical therapies.

Many methods may be considered to pursue these tasks, each with its own characteristics that are of specific interest according to the application we are considering.

Logic models of the type considered in this paper are of particular relevance when the search for a good *explanatory model* is important at least as the precision of the classification method, as they extract *logic rules* that are easy to understand and to integrate with additional knowledge.

Many different approaches have been proposed in the literature for the task of extracting classification models from data in logic form. We briefly mention the seminal work on Classification Trees method (Breiman et al. 1984), whose basic methodology has been evolved and consolidated in many data analysis tools largely used in applications; the very exhaustive work on the LAD methodology (Boros, Ibaraki, and Makino 1999), (Boros et al. 1997),(Boros et al. 2000), characterized by a deep understanding of the mathematical and combinatorial roots of this problem; the work of (Triantaphyllou and Soyster 1996).

The methodology discusses in this paper is based on previous work, among which (Felici and Truemper 2002), (Truemper 2004), (Felici and Truemper 2005), (Felici, Sun, and Truemper 2006), (Bertolazzi, Felici, and Lancia 2010). This latter approach has proved particularly well suited for biological data analysis; successful applications in this area are described in (Bertolazzi et al. 2008), (Bertolazzi, Felici, and Weitschek 2009), (Arisi et al. 2011).

This approach has been recently organized into a collection of software tools, the DMB system (Data Mining Big), available for use on the web (dmb.iasi.cnr.it), where the logic data mining process is organized in three main steps, each relying on optimization models and algorithms: *Discretization*, *Feature Selection*, and *Formula Extraction*.

In the following sections we will consider several issues related with the presence of noisy data that specifically arise in these three steps. Before going into greater detail, we find beneficial to introduce a basic notation that will be shared by the different sections and extended when needed.

The basic format of the input data is a a set T of n elements and a collection of m features $\{f_1, \ldots, f_m\}$, each representing the value of a measure over the elements. We thus indicate with e_i the i^{th} element in T and with f_{ij} the value of feature j on element e_i.

The set T is divided into two classes, and we look for a model that explains the separation of set T in these two classes, based on the values of the features.

We indicate the two classes with *class A* and *class B*; The results described below, if not else indicated, can be straightforwardly extended to the case of more than 2 classes. An additional and required assumption is that the two classes do not intersect, that is, there are no elements with the same values for all the features that belong to different classes. If this case should arise, it is assumed that such elements are removed from set T or ignored in the learning process.

For each item i, each feature f_j takes a value in a given scale. In the standard setting, each feature is of logic type and has two possible values (*True* and *False*) representing the presence or the absence of a given characteristic, associated with that feature, in element e_i.

In general, features may represent measures over a real or discrete scale, and in order to adopt a logic approach we need to properly convert such features with a *discretization* procedure that converts a non logic feature into one or more logic ones, each associated with an interval of the scale of the original feature.

The number of features that are to be taken into consideration for a real application could be extremely large, easily rising to many thousands, as is typically the case when dealing with biological or genetic data. Such dimensions result prohibitive for most knowledge extraction methods, and it is therefore advisable to identify the features that are of interest for the classification model with a specific method, specifically designed and made efficient for that purpose: this is the *feature selection* step.

When a reduced and promising number of binary or logic features has been identified, we can then proceed with the third and last step, *formula extraction*, that usually requires accurate and computationally demanding algorithms.

Noise Reduction in Discretization

When some of the variables are represented by integer or real numbers we need to apply a transformation and convert each such variable into a new discrete variable that is suitable for treatment into any logic framework. This step amounts in determining a set of *cutpoints* over the range of values that each variable may assume and thus define a number of intervals over which the original variable may be considered discrete.

Here we present a new method based on recent experiments, designed for properly dealing with cases where a certain amount of noise may be present in the data.

When intervals are defined over a continuous scale, we are interested in their purity; an interval is considered pure if all the elements that have that feature in that interval belong to the same class. The situation where all intervals for all features are pure may result in an excessive requirement: as a matter of fact, a separation among the classes can be achieved by having one pure interval for each element, or by the combination of more non-pure intervals. Given an interval, we then adopt as measure of its *purity* the normalized reciprocal of the class entropy of the elements that belong to that interval; as detailed later, we aim for a discretization where each element of the training set receives at least one interval with a good value of the purity measure.

Previous versions of the method were based on the iterative maximization of the purity of each interval; here we consider the additional problem of noisy data and its potential disturbance on the discretization process. A strict requirement on the purity of the intervals may in fact lead to the identification of too many small intervals; the problem then amounts in determining a proper balance between the number of intervals (that is to be minimized) and the purity of the intervals (that is to be maximized). Clearly, these two objectives are in contrast: the purity of the intervals cannot increase with the reduction of the number of intervals, and vice versa.

We define below a formal framework to deal with this problem.

We start by considering the projection of the data onto a single feature, say feature k, and f_{ik} for $i = 1, ..., n$ as the vector of the values that each element e_i assumes for feature k. In order to define the intervals for discretization we use *cutpoints* in the scale of the feature, e.g., values that determines the extreme of the intervals. Without loss of generality we assume that vector f_{ik} is in non-decreasing order and we mark the positions where a change in the class of two consecutive elements occurs. In these positions are the cutpoints that define only *pure intervals*. More formally, we define as nc_k the number of cutpoints for feature k, and with $c^k = \left\{c_1^k, ..., c_{nc_k}^k\right\}$ the values of such cutpoints, where $c_j^k = 0.5 \times (f_{ik} + f_{i+1,k})$ whenever elements i and $i+1$ belong to different classes.

Now we build a measure of the discriminating power of a cutpoint. In order to do this for a given cutpoint c_j^k, we call a *crossing* any pair of elements (i_1, i_2) that belong to different classes and lie on different sides of the cutpoint. With $NC(c_j^k)$ we indicate the set of all crossings of c_j^k. Then, for each crossing (i_1, i_2) we define a *distance* $d(i_1, i_2)$ equal to the number of position that occur between i_1 and i_2 when the non decreasing order of the f_{ik} is considered. Finally, for a real parameter λ_k, we compute the discriminating power of cutpoint c_j^k as

$$DP(c_j^k, \lambda_k) = \sum_{(i_1,i_2) \in NC(c_j^k)} 1 \times e^{-\lambda_k \times d(i_1,i_2)}$$

$DP(c_j^k, \lambda_k)$ measures how good is a given cutpoint in discriminating elements of different classes. The parameter λ_k plays an important role in this measure: when $\lambda_k = 0$ each crossing has the same weight and therefore in the computation there is no account of *locality*: one single cutpoint will receive the largest value of $DP()$, and this cutpoint would be the best choice if we decide to choose only one cutpoint. Else, when λ_k becomes large, the measure becomes more local, and the value of $DP(c_j^k, \lambda_k)$ tends to be the same for all the cutpoints of the same feature.

Given a value λ_k, for obvious reasons we propose to choose as candidate cutpoints those that are associated with peaks in the values of the discriminating power. Below we discuss a sensible rule to pick the value of λ_k for all k and motivate its beneficial role in the presence of noisy data.

To do so, we now turn to consider all the m features. For each feature k it is easy to know (e.g. with binary search) the smallest value of λ_k for which all the cutpoints of k receive the same discriminating power. Call this value λ_k^{max}. Define now a m-dimensional vector $\lambda = \{\lambda_1, \lambda_2, \dots, \lambda_m\}$ where, for each k, $0 \leq \lambda_k \leq \lambda_k^{max}$.

For a given multidimensional value of vector λ, we can compute:

- the total number of selected cutpoints over all the features (recall that a cutpoint is selected when its discriminating power is a local maximum with respect to its neighboring cutpoints); we refer to the total number of cutpoints associated with λ as $CP(\lambda)$;
- the largest purity value that is assigned to at least one interval for each element when the cutpoints are selected as above; we refer to this value as $NPI(\lambda)$ (note that this value is the lower bound on the highest purity value of the intervals over all the elements of the training set).

From the values above we can compute two reference values: $\lambda^{min} = \{0, 0, \dots, 0\}$ and $\lambda^{max} = \{\lambda_1^{max}, \lambda_2^{max}, \dots, \lambda_m^{max}\}$ that are the two values of the k-dimensional vector λ to which correspond the extremes in the number of cutpoints $CP(\lambda)$ and in the lower bound on pure intervals $NPI(\lambda)$.

It is easy to show that:

$$CP(\lambda_{min}) = k; CP(\lambda_{max}) = \sum_k nc_k$$

and

$$NPI(\lambda_{min}) \geq 0; NPI(\lambda_{max}) = 1$$

Having at hand the range of variation for $CP(\lambda)$ and $NPI(\lambda)$ we can normalize their value to obtain a relative measure defined as below:

- *Robustness:* $R(\lambda) = \frac{(CP(\lambda^{max}) - CP(\lambda))}{(CP(\lambda^{max}) - CP(\lambda^{min}))}$;
- *Precision:* $P(\lambda) = \frac{(NPI(\lambda^{max}) - NPI(\lambda))}{(NPI(\lambda^{max}) - NPI(\lambda^{min}))}$;

we define the *discretization quality* (DQ) as the product of robustness and precision:

$$DQ(\lambda) = R(\lambda) \times P(\lambda)$$

and adopt it as a final measure of the quality of the discretization associated with a k-dimensional vector λ. We note that this measures increases monotonically both with the robustness and the precision of the discretization, and its maximization provides a very reasonable balancing of these two criteria.

The problem then amounts to solve the following optimization problem:

$$max_{(\lambda^{min} \leq \lambda \leq \lambda^{max})} DQ(\lambda).$$

The identification of a good solution for the problem above is determined by a search procedure over the k-dimensional compact sub space identified by $\lambda^{min} \leq \lambda \leq \lambda^{max}$. We choose to adopt λ^{max} (minimum robustness, maximum purity) as a starting point and gradually decrease λ component by component (see Figure 1):

```
Discretization
1:  for every numeric feature $f_i$
2:      compute $nc_k$ and $\lambda_k^{max}$
3:  end for
4:  compute $R(\lambda^{max})$, $P(lambda^{max})$, $DQ(\lambda^{max})$
5:  set $DQ_0 = 0, DQ_1 = DQ(\lambda^{max}), i = 1$
6:  while $(DQ_i > DQ_{i-1})$
7:      select $k$ and $\delta$, update $\lambda_k = max(0, \lambda_k - \delta))$
8:      $i = i + 1$; compute $DQ_i = DQ(\lambda)$
9:  end while
10: output cutpoints associated with $\lambda$
```

Figure 1: Discretization in DMB

The main step of the procedure described in Figure 1 is clearly step 7, where we select one of the features and decrease its λ value; this amounts to reducing the number of cutpoints for feature k and thus increasing robustness and decreasing precision. We implement this step in a very straight forward fashion, adopting a greedy strategy: for each feature we select the value of δ that would imply the loss of only one cutpoint; then, we compute the related loss in precision implied by the loss of this cutpoint. The cutpoints that score the smallest loss in precision are candidate for the choice and we select among them at random. When this strategy is not able to improve the current solution we stop the search in a local minimum.

Once the procedure terminates, we are left with a possibly small number of cutpoints that can be used to discretize the features, and we can represent the discretization with a number of *True-False* or binary features associated with each interval. The procedure is designed in such a way that those features affected by noise would receive a less granular discretization: in a feature affected by noise the cutpoints will be very close to each other and will have a limited discriminating power; for this reason, they will be eliminated as their contribution to precision will be small as opposed to other cutpoints.

Noise Reduction in Feature Selection

When dealing with binary features, the problem of selecting a subset of features of minimal size that guarantees the separation between two sets can be formulated as an Integer Linear Programming Problem, more specifically as a variant of the Set Covering problem.

A basic definition of the problem of feature selection (called *test cover*) is presented in (Garey and Johnson 1979).

We say that a feature f_j differentiates (covers) item pair $\{k, h\}$ if $f_{jk} \neq f_{jh}$. If we consider all the pairs of items $\{k, h\}$ where k and h belong to different classes, then a subcollection $\mathcal{F} \subset \mathbf{F}$ of features is a cover if each of such pairs $\{k, h\}$ are covered by at least one element in $\mathcal{F}$. We note that the number of pairs to be considered is equal to the product of the cardinalities of the two classes, that grows quadratically with n. The problem of finding a subset of features of minimal size that covers all the pairs of distinct elements is called *Combinatorial Feature Set* or *Minimal Test Collection*. A mathematical formulation of the problem is given below:

$$\begin{array}{ll} \min & \displaystyle\sum_{j=1}^{m} x_j \\ & \displaystyle\sum_{j=1}^{m} a_{ij} x_j \geq 1 \qquad i = 1 \ldots M \\ & x_j \in \{0,1\} \qquad j = 1 \ldots n, \end{array} \tag{1}$$

where $x_j = 1$ if f_j is chosen and 0 otherwise; each of the M constraints is associated with a pair of items belonging to different classes; e.g., if row i is associated with the item pair $\{k, h\}$ then we have that, for feature j, $a_{ij} = 1$ if and only if $f_{jk} \neq f_{jh}$.

In (Bontridder et al. 2002) is presented a branch-and-bound procedure based on a new definition of branching rules and lower bounds for the above problem. Nevertheless, when the problem size is large, the use of optimization algorithms to produce guaranteed optimal solutions becomes impractical, and one has to resort to heuristics schemes.

The above approach to the FS problem presents also additional drawbacks. Its purpose is to find a minimal set that can separate the given data, that plays the role of *training* data in the general process. The features associated with the minimal set are then used to project the training data and to derive classification rules, that are then applied to *test* data, once the latter has been projected using the same feature set.

When the data is noisy or not well sampled, it may happen that the effect of good features - e.g., features that should belong to the final model - is masked by the noise in some data and the thrifty representation adopted with the previous model does not leave space to fix this error; for this reason, maintaining a certain measure of redundancy in the information that is retained by the FS selection results into good strategy for the production of rules with good predictive power in the presence of noisy data.

This is particularly true when FS is followed by classification algorithms that can perform an additional selection of the features based on the separating model. For this reason, we want to consider cases where the *rhs* of the covering constraints is greater than 1. This would result in the selection of a larger set that brings more information to the formula extraction step; more precisely, we propose to adopt an optimization model where the number of features to be selected is fixed in advance by the parameter β, and the level of redundancy α is maximized in the objective function:

$$\begin{array}{ll} \max & \alpha \\ & \displaystyle\sum_{j=1}^{m} a_{ij} x_j - \alpha \geq 0 \quad i = 1 \ldots M \\ & \displaystyle\sum_{j=1}^{m} x_j \leq \beta \\ & x_j \in \{0,1\} \qquad j = 1 \ldots m, \end{array} \tag{2}$$

To solve the generalized set covering problems we pursue a non-deterministic and heuristic method known in literature as GRASP, acronym of Greedy Randomized Adaptive Search Procedure, proposed in Feo and Resende (Feo and Resende 1989; 1995) for which extensive details are made available in the annotated bibliographies (Resende and Ribeiro 2002) and (Festa and Resende 2009). GRASP is a randomized heuristic based on a construction phase and a local search phase. The first adds one feature at a time to a set until a representation of a feasible solution is obtained: the feature is randomly selected from a *restricted candidate list* RCL, whose elements are those that obtain the top scores according to some greedy function. Once a feasible solution is obtained, the local search procedure attempts to improve it by producing a locally optimal solution with respect to some neighborhood structure.

The construction and the local search phases are applied repeatedly, finally returning the best solution found. A detailed description of the implementation details of this solution method goes beyond the scope of this paper and can be found in (Bertolazzi et al. 2008) and related papers.

The proposed feature selection method is designed to achieve robust results at the price of a certain level of redundancy, in order to absorb the potential bias on the results induced by noisy data. The quality of the results would although depend on the choice of the parameter β. An additional refinement of the method takes into account a strategy for the right choice of β, based on a more extensive analysis of the relations between the values of α and β.

We first note that the value of the solution, α, cannot decrease when the parameter β is increased; therefore, we solve a initial instance of the problem with a sufficiently large value of β_0, obtaining a corresponding solution $\alpha_0 > 0$. Then, we reduce β gradually and obtain corresponding α values, until we reach the smallest value β_h for which $\alpha = 1$; from these runs we derive the values of β needed to obtain $\alpha = 0, 1, 2, \ldots, \alpha_0$. We can now inspect these values and measure the increase in β that is needed to obtain an increase of 1 in the value of α. For ease of reference, call β_h the value of β needed to obtain $\alpha = h$, and call $\hat{\beta}_h = \beta_h - \beta_{h-1}$; then, when $\hat{\beta}_{h+1} >> \hat{\beta}_h$ we stick to the value β_h, and adopt the corresponding solution of value h to identify the features to be selected and passed on to the next step.

Noise Reduction in the Extraction of Logic Formulas

The learning tool used in this application, *Lsquare*, is a particular learning method that operates on data represented by logic variables and produces rules in propositional logic that classify the items in one of two classes. *Lsquare* adopts a logic representation of the description variables, that are to all extents logic variables, and of the classification rules, that are logic formulas in Disjunctive Normal From (DNF).

The *Lsquare* system and some of its additional components have been described in other work ((Felici and Truemper 2002), (Felici and Truemper 2005), (Truemper 2004)) and its detailed description is out of the scope of this paper. Here, we simply mention some of its distinctive features, and describe a variant of the method designed to refine the solution in order to spot the presence of noise in the data and to mitigate its potentially negative effects.

Lsquare is based on a particular problem formulation that amounts to be a well know and hard combinatorial optimization problem, the *minimum cost satisfiability problem*, or MINSAT, that is solved using a sophisticated solver based on decomposition and learning techniques (Truemper 2004).

The identified DNF formulas have the property of being created by conjunctive clauses that are searched for in order of coverage of the training set. Therefore, they usually are formed by few clauses that explain a large portion of the data (clauses with large coverage, that synthesize the interpretation of the trends present in the data) and several clauses that explains few of the examples in the training set (smaller coverage: the interpretation of the outliers).

At this stage of the process we may assume that part of the noise may have been eliminated in the previous steps; nevertheless, being the method exposed to overfitting by its very nature, we adopt an iterative pruning approach that refines the formulas getting rid of variables that appear in the model but do not show a relevant contribution.

We recall that an explanatory model determined by *Lsquare* is composed of a logic formula in DNF, referred to as F, designed to have value *True* on one of the classes (say, class A) and value *False* on the other one (class B; the role of the classes could be inverted if needed). Such a formula is composed of a number of conjunctive clauses, combined with disjunctions. Each clause, in turn, is composed of a finite number of literals, e.g., occurrence of a logic variable or its negation. We indicate a logic variable that appears in F as v_i.

Once F has been determined, we define a pruning procedure that removes a variable at a time according to the rules described below, until a certain stopping criterion is met.

First we need to define, for every logic variable v_i that is present in F, three measures:

- $ex(v_i)$, or *exclusive coverage* of v_i: the number of couples of training elements belonging to different classes that are differentiated only by v_i in F;
- $sc(v_i)$, or *score* of v_i: we eliminate the variable v_i from the formula F and then evaluate the reduced formula on training data. Such evaluation will result in number of errors (false negatives $fn(v_i)$ and false positives $fp(v_i)$). The score $sc(v_i)$ is exactly the number of such errors;
- $ds(v_i)$, or *decreasing speed* of v_i: it is defined as the difference between the score obtained by v_i at the previous iteration and the one obtained at the current one.

These measures are the main ingredients of the proposed pruning procedure. When $ex(v_i)$ is sufficiently large we know that the v_i cannot be substituted by another one, and we are not inclined to prune it. The score measure $sc(v_i)$ tells with higher precision what would happen if we remove this variable.

At each iteration, we select the variable with minimum score, breaking ties arbitrarily; assume v_j to be such variable.

Then, we consider its exclusive coverage $ex(v_j)$; if $ex(v_j) \leq 0.05 * n_A * n_B$ (with n_A and n_B we indicate the number of elements in classes A and B, respectively) we allow to prune v_j knowing that at most 5% of the couples of elements of different class may not be correctly distinguished when a model without v_j is calculated by *Lsquare*.

A special role is assigned to the the decreasing speed $ds(v_i)$. Features whose score tends to decreases with the pruning iterations are in fact less important that those for which the score increases; as fewer features are left at each iteration, we know that the contribution of good features will increase, and vice versa. The decreasing speed captures this behavior: when high, it indicates that a feature is eligible for pruning; when small, it indicates that the feature is to be kept. We use the $ds(v_i)$ to determine a stopping criterion: if no features have $ds() > 0$ we interrupt the pruning.

The procedure is synthesized in the scheme of Figure 2:

Conclusions

We performed several experiments injecting a controlled amount of noise in real data sets from the literature or simulating ad hoc noisy data sets. In particular, we tested these new variants of DMB on the data provided by Thorsten Lehr and described in (Lehr et al. 2011). We witness a very good behaviour of the system in getting rid of different overfitting effects due to the introduction of noise.

We conclude with two additional observations.

The proposed methods appear to be very robust and stable with respect to different data sets, and they do not rely on

```
Pruning the Logic Model
0: while
1:   for all logic variables $v_i$ compute $ex(v_i)$, $sc(v_i)$, $ds(v_i)$
2:   end for
3:   select $v_j$ such that $sc(v_j)$ is minimum;
4:   $if(ex(v_j) \leq 0.05 * n_A * n_B)$
5:     $AND$ (there exists $v_k$ such that $ds(v_k) > 0$)
6:     remove $v_j$ from $F$
7:     run Lsquare;
8:   $else$ break
9: end while
```

Figure 2: Pruning procedure

the use of verification data to tune parameters or evaluate the effect of the pruning; the refinements proposed for error control rely solely on the analysis of the training set.

The methods do require an additional computational effort, as in the three phases the problem need to be solved several times with different parameters; although, such effort is limited and can be assumed be bounded by a small multiple of the effort needed to compute a single iteration. Proper implementation of the discretization and feature selection steps would anyway allow to run the new procedure without additional effort. In the formula extraction phase the cost of running the algorithm several time has to be taken into account, although such cost is typically contained, as the problem has already been drastically reduced in size by the two previous steps.

References

Arisi, I.; D'Onofrio, M.; Brandi, R.; Felsani, A.; Capsoni, S.; Drovandi, G.; Felici, G.; Weitschek, E.; Bertolazzi, P.; and Cattaneo, A. 2011. Gene expression biomarkers in the brain of a mouse model for alzheimer's disease: Mining of microarray data by logic classification and feature selection. *Journal of Alzheimer's Disease* 24(4):721–738.

Bertolazzi, P.; Felici, G.; Festa, P.; and Lancia, G. 2008. Logic classification and feature selection for biomedical data. *Comput Math Appl* 55:889–899.

Bertolazzi, P.; Felici, G.; and Lancia, G. 2010. Application of feature selection and classification to computational molecular biology. In J.K. Chen, S. L. e., ed., *Biological Data Mining*. Chapman & Hall. 257–294.

Bertolazzi, P.; Felici, G.; and Weitschek, E. 2009. Learning to classify species with barcodes. *BMC Bioinformatics* 10(Suppl 14):S7.

Bontridder, K. M. J. D.; Lageweg, B. J.; Lenstra, J. K.; Orlin, J. B.; and Stougie, L. 2002. Branch-and-bound algorithms for the test cover problem. In Mhring, R. H., and Raman, R., eds., *ESA*, volume 2461 of *Lecture Notes in Computer Science*, 223–233. Springer.

Boros, E.; Hammer, P.; Ibaraki, T.; and Kogan, A. 1997. Logical analysis of numerical data. *Mathematical Programming* (79):163–197.

Boros, E.; Hammer, P.; Ibaraki, T.; Kogan, A.; and Mayoraz, E. M. I. 2000. An implementation of logical analysis of data. *IEEE Transactions on Knowledge and Data Engineering* 2(12):292–306.

Boros, E.; Ibaraki, T.; and Makino, K. 1999. Logical analysis of binary data with missing bits. *Artif. Intell.* 107:219–263.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Statistics/Probability Series. Belmont, California, U.S.A.: Wadsworth Publishing Company.

Felici, G., and Truemper, K. 2002. A minsat approach for learning in logic domains. *INFORMS Journal on computing* 14:20–36.

Felici, G., and Truemper, K. 2005. *The Lsquare System for Mining Logic Data*. Idea Group Publishing.

Felici, G.; Sun, F.; and Truemper, K. 2006. Learning logic formulas and related error distributions. In Felici, G., and eds., E. T., eds., *Data Mining and Knowledge Discovery Approaches Based on Rule Induction Techniques*. Springer.

Feo, T., and Resende, M. 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* 8:67–71.

Feo, T., and Resende, M. 1995. Greedy randomized adaptive search procedures. *J. of Global Optimization* 6:109–133.

Festa, P., and Resende, M. 2009. An annotated bibliography of GRASP – Part I: Algorithms. *International Transactions in Operational Research* 16(1):1–24.

Garey, M., and Johnson, D. 1979. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W.H. Freeman.

Lehr, T.; Yuan, J.; Zeumer, D.; Jayadev, S.; and Ritchie, M. 2011. Rule based classifier for the analysis of gene-gene and gene-environment interactions in genetic association studies. *BioData Mining* 4(1):4.

Resende, M., and Ribeiro, C. C. 2002. Greedy randomized adaptive search procedures. *Handbook of Metaheuristics* 219–249.

Triantaphyllou, E., and Soyster, A. L. 1996. On the minimum number of logical clauses inferred from examples. *Computers & Operations Research* 23(8):783–799.

Truemper, K. 2004. *Design of Logic based Intelligent Systems*. Wiley Interscience.